©2002 Microsoft Corporation

Using Data Access in .NET Applications

Agenda: Day 1

Module 01: What is Microsoft® .NET?

Module 02: Visual Studio® .NET

Module 03: The .NET Languages

Module 04: Building ASP® .NET Applications

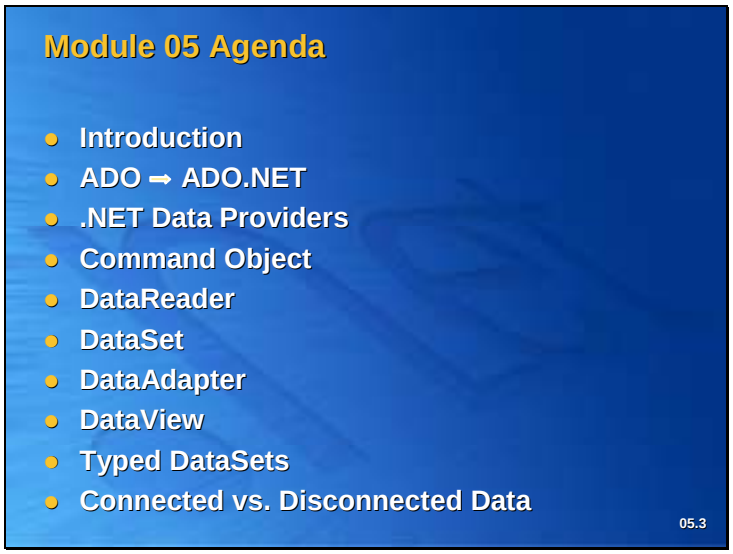
▶ Module 05: Using Data Access in .NET Applications

Module 06: Object Oriented Concepts in .NET

Module 07: Creating Windows Applications

Module 08: Using XML in .NET

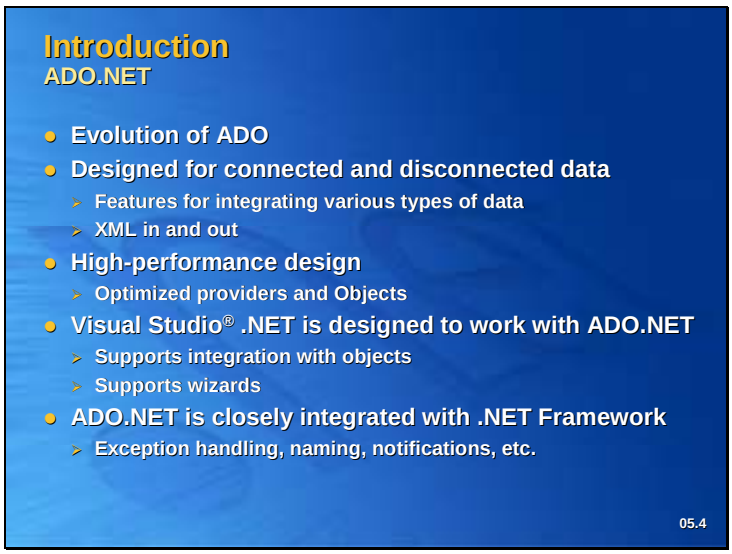
 ©2002 Microsoft Corporation



Module 05 Agenda

What you will learn:

- How ADO.NET is different from ADO
- The benefits of ADO.NET
- ADO.NET Core Concepts and architecture including the DataSet, data views, and connected versus disconnected data
- How the new ADO.NET DataReader, DataSet, and DataAdapter work


Introduction
ADO.NET
Evolution of ADO
Designed for connected and disconnected data
Features for integrating various types of data
XML in and out
High-performance design
Optimized providers and Objects
Visual Studio® .NET is designed to work with ADO.NET
Supports integration with objects
Supports wizards
ADO.NET is closely integrated with .NET Framework
Exception handling, naming, notifications, etc.
05.4

Introduction

ADO.NET

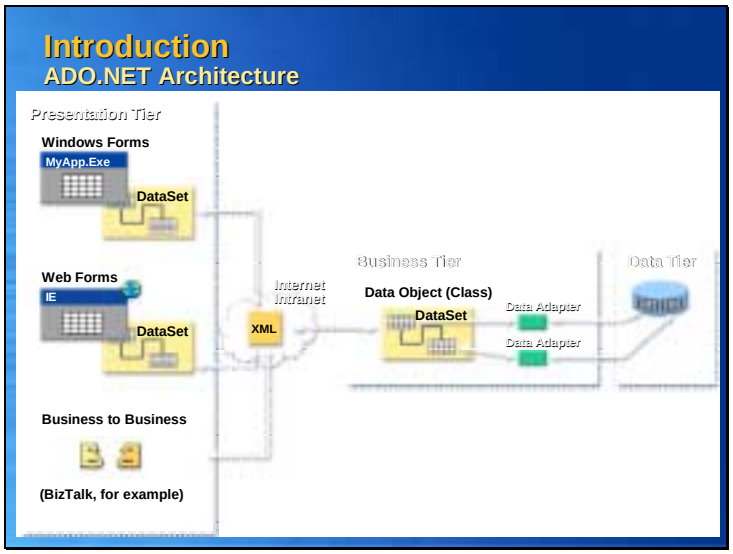
An improvement to Microsoft® ActiveX® Data Objects, **ADO.NET** provides a platform interoperability and scalable data access. Because Extensible Markup Language (XML) is the format for transmitting data, any application that can read data in the XML format can process data. In fact, the receiving component need not be an ADO.NET component. It can be a Microsoft Visual Studio-based solution or any application operating on any platform.

Visual Studio .NET gives you the opportunity to program against your objects, not against tables and columns. ADO.NET uses strongly typed programming in which business objects figure prominently.

The centerpiece of any software using ADO.NET is the DataSet, an in-memory copy of database data that contains any number of datatables each typically corresponding to a database table or view. A DataSet constitutes a disconnected view of the database data, or it exists in memory without an active connection to a database containing the corresponding tables or views. This disconnected architecture enables greater scalability by using only database server resources when reading or writing from the database.

At run time, data is passed from the database to a middle-tier business object and then down to the user interface. To accommodate the exchange of data, ADO.NET uses an XML-based persistence and transmission format. That is, to transmit data from one tier to another, an ADO.NET solution expresses the in-memory data (the DataSet) as XML and sends the XML to the other component.

 ©2002 Microsoft Corporation

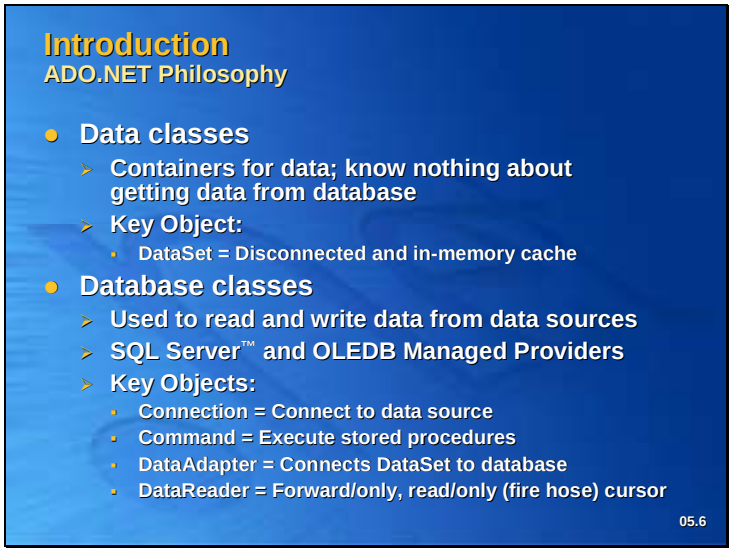


Introduction

ADO.NET Architecture

The illustration on the slide shows the major components of an ADO.NET solution.

It includes new features that make working with data easy in Visual Studio .NET. For the XML developer, Microsoft has included a rich, color-coded XML Designer with statement completion and tag completion.


Introduction
ADO.NET Philosophy
Data classes
Containers for data; know nothing about getting data from database
Key Object:
DataSet = Disconnected and in-memory cache
Database classes
Used to read and write data from data sources
SQL Server™ and OLEDB Managed Providers
Key Objects:
Connection = Connect to data source
Command = Execute stored procedures
DataAdapter = Connects DataSet to database
DataReader = Forward/only, read/only (fire hose) cursor
05.6

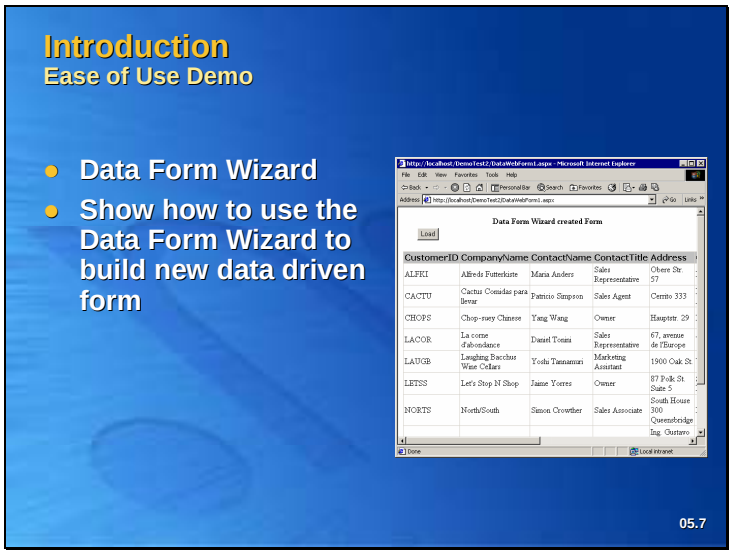
Introduction

ADO.NET Philosophy

ADO.NET data classes are used when working with disconnected data.

In ADO.NET, there are four classes that read and write data from data sources: the Connection, Command, DataAdapter and DataReader classes.

 ©2002 Microsoft Corporation


Introduction
Ease of Use Demo
Data Form Wizard
Show how to use the Data Form Wizard to build new data driven form
Data Form Wizard created Form
Load
CustomerID CompanyName ContactName ContactTitle Address
ALFKI Alfreds Futterkiste Maria Anders Sales Representative Obere Str. 57
CACTU Cactus Comidas para llevar Patricio Simpson Sales Agent Cerrito 333
CHOPS Chop-suey Chinese Yang Wang Owner Hauptstr. 29
LACOR La corne d'abondance Daniel Tonini Sales Representative 67, avenue de l'Europe
LAUGB Laughing Bacchus Wine Cellars Yoshi Tannamuri Marketing Assistant 1900 Oak St.
LETSS Let's Stop N Shop Jaime Yorres Owner 87 Polk St. Suite 5
NORTS North/South Simon Crowther Sales Associate South House 300 Queensbridge
Ing. Gustavo
05.7

Introduction

Ease of Use Demo

The Data Form Wizard automatically generates forms that contain individual bound controls and procedures used to manage information from database tables and queries. The Wizard can create either single query forms that manage the data from a single table or simple query or Master/Detail Forms that manage more complex one-to-many data relationships. If using a control, you also can create a grid or datasheet form.

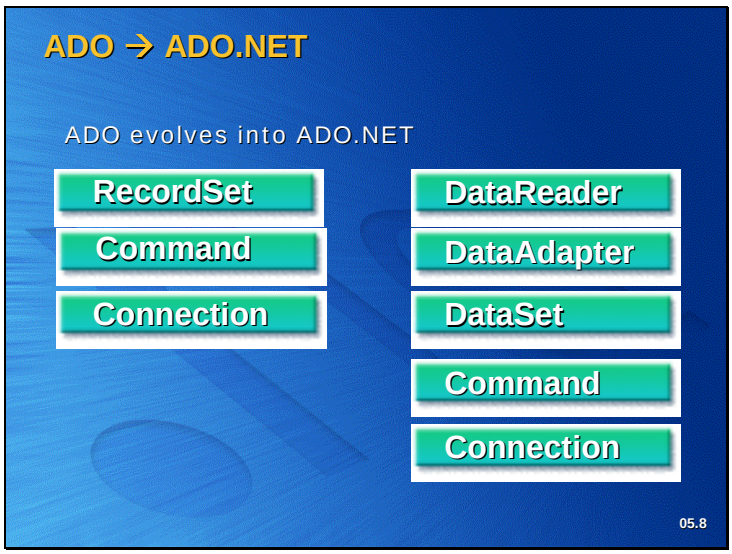
Use the Data Form Wizard to rapidly build forms with controls bound to a data source, create single record, grid, and Master/Detail Forms or to rapidly create prototype data-based forms.

Data Form Wizard

Show how to use the Data Form Wizard to build new data driven form.

The purpose of this demo is to show:

- Options when creating form
- Form code generated



ADO → ADO.NET

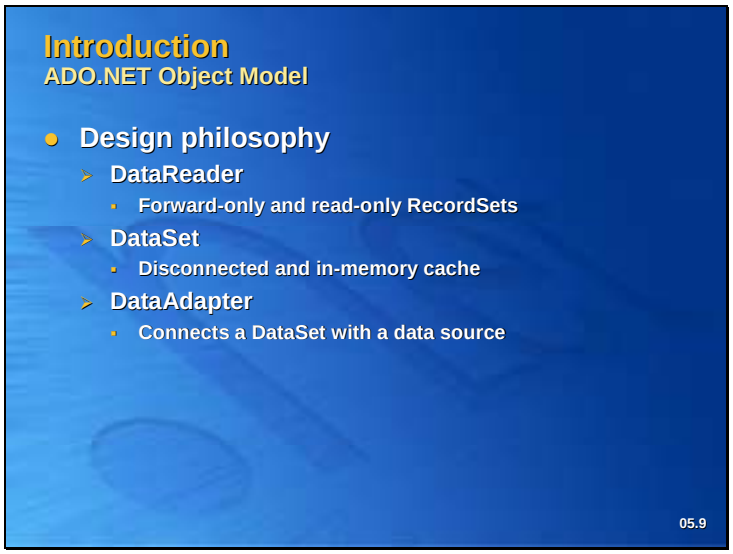
DataReader vs. RecordSet

- RecordSet uses special programming models while DataReader uses stream interface
- RecordSet returns fields as variants while DataReader returns fields as type-safe datatypes for faster code. You also can get them as objects
- DataReader is the fastest way to access data today

Why factor out these objects?

- The predictability of the RecordSet was uncertain and new objects were factored to allow them to always be predictable
- The performance is better because each object is designed for a specific type of access
- The programming model is designed to work with streams of data more efficiently

 ©2002 Microsoft Corporation



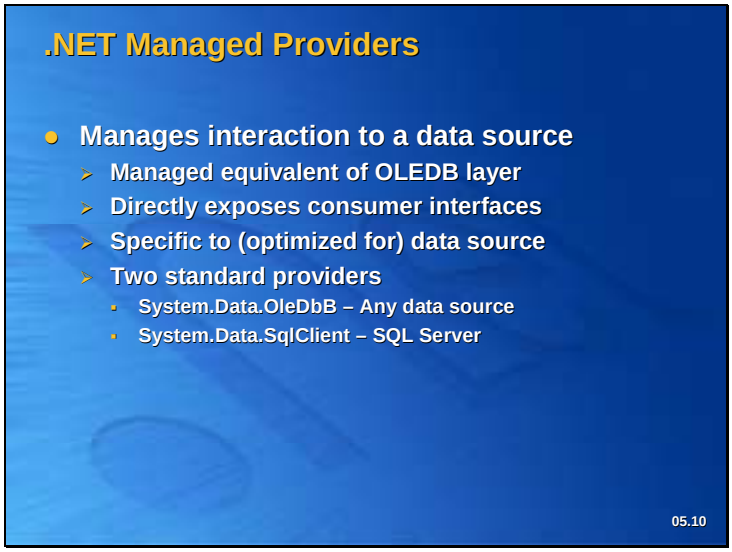
Introduction

ADO.NET Object Model

ADO, as it exists today and ADO RecordSets are useful tools for manipulating data in a Windows and COM-based scenario. Unfortunately, they progressively lose their appeal when a system evolves in the direction of total Internet interoperability. For example, XML Web services do not accept ADO RecordSets. **In ADO.NET, the ADO RecordSet has been "factored" into the DataReader, the DataSet and the DataAdapter.**

An ADO RecordSet is a massive object with a variety of positive aspects in that it is creatable, works disconnected and is feature-rich. Improvements could be made, however, in some areas. For example, ADO RecordSets are heavy to serialize over a network because of its inherent COM nature, it's a binary object that is difficult to share between modules operating on different platforms and it is unable to penetrate firewalls. In addition, it represents a single table of records. If this table comes as the result of one or more JOINs, it can be difficult to update the original data sources. When attempting to reconcile the disconnected RecordSets with the original data source, it works as long as the data source understands SQL. The RecordSets easily could have been created, however, by a non-SQL provider.

In ADO.NET, the functionality of the ADO RecordSet is split into a few simpler objects which are faster to transport and more efficient.


.NET Managed Providers
Manages interaction to a data source
Managed equivalent of OLEDB layer
Directly exposes consumer interfaces
Specific to (optimized for) data source
Two standard providers
System.Data.OleDbB – Any data source
System.Data.SqlClient – SQL Server
05.10

.NET Managed Providers

Managed providers are comprised of a collection of classes for accessing various data sources. Included in the .NET Framework you will find classes specifically optimized for Microsoft SQL Server 2000, SQL Server 7, and Microsoft Data Engine (MSDE). Of course, there is also a host of other classes for working with any OLE DB providers, including: Oracle, Jet, and SQL OLE DB providers. The following drivers have been tested for ADO.NET:

- SQLOLEDB, Microsoft OLE DB Provider for SQL Server
- MSDAORA, Microsoft OLE DB Provider for Oracle
- Microsoft.Jet.OLEDB.4.0, OLE DB Provider for Microsoft Jet

The OLE DB .Net providers can not be used to access data from ODBC data source. To access an ODBC data source using ADO.NET, use the ODBC .NET Data Provider, this is available for download at http://msdn.microsoft.com/downloads.

As the term *managed providers* implies, these classes are designed to provide connectivity to the data source; they are *the* connection between the **DataSet** object and the data store.

Two Managed Providers

In the .NET Framework there are currently two managed providers:

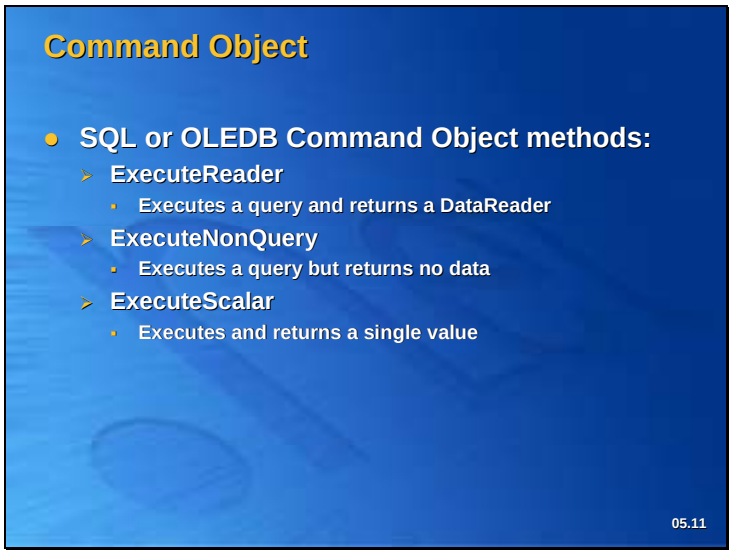
- OLE DB: via the **System.Data.OleDb** namespace
- SQL Server: via the **System.Data.SqlClient** namespace

The latter is optimized for SQL Server; it is the SQL Server managed provider.

System.Data.ADO is* the *ADO.NET managed provider

The **System.Data.ADO** namespace is the ADO.NET managed provider.

 ©2002 Microsoft Corporation



Command Object

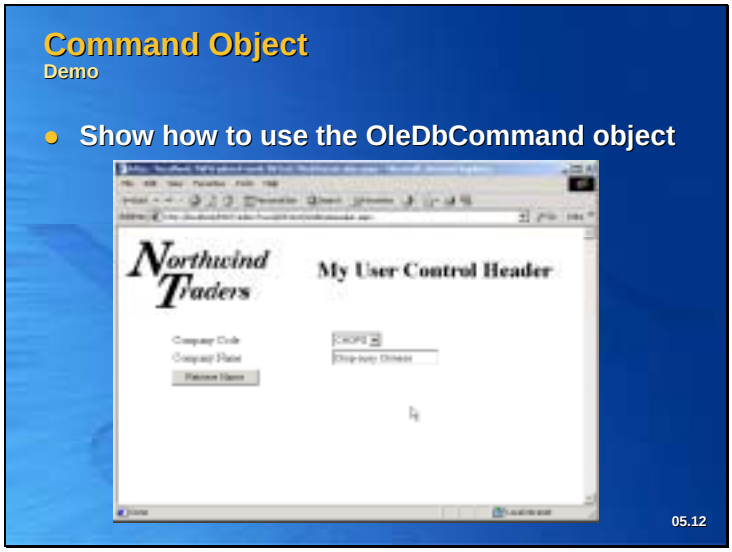
The Command object represents a specific command given against an OLEDB data source. The Command object has three execute methods: ExecuteReader, ExecuteNonQuery, and Execute Scalar.

The ExecuteReader method executes a SQL query or a stored procedure and returns a DataReader. The ExecuteNonQuery method executes a SQL query or a stored procedure and returns no records.

The ExecuteScalar method can be used to retrieve a single value (for example, an aggregate value) from a database. This requires less code than using the ExecuteReader method and performs the operations necessary to generate the single value using the data returned by a SqlDataReader. A typical ExecuteScalar query can be formatted as in the following Visual C# example:

```
SqlCommand cmd = new SqlCommand;
cmd.CommandText = “Select count(*) as NumberOfRegions from region";
int count = (int) cmd.ExecuteScalar();
```

For more information about the OleDbCommand and SqlCommand classes search for the keywords “OleDbCommand” and “SqlCommand” in the Visual Studio .NET online help.

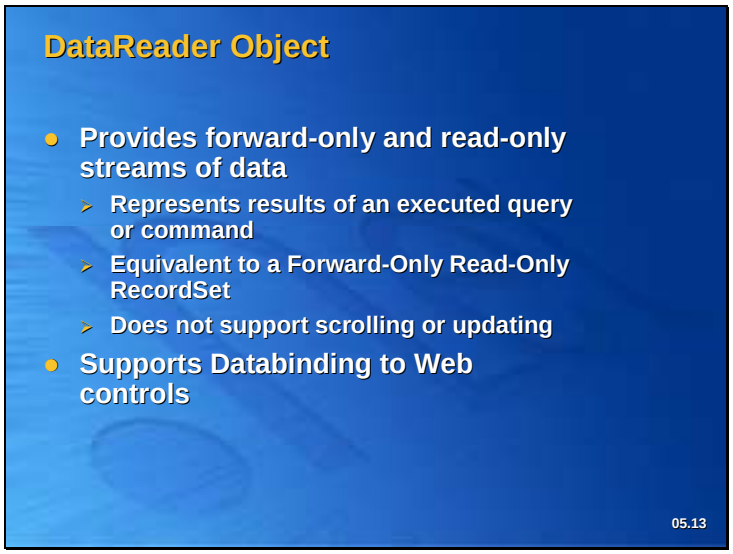


Command Object Demo

The purpose of this demo is to show how to use the OleDbCommand object:

- How command object is part of OLEDB and SqlClient providers?
- How to use ExecuteScalar method?
- Simple structure of using command object

 ©2002 Microsoft Corporation


DataReader Object
Provides forward-only and read-only streams of data
Represents results of an executed query or command
Equivalent to a Forward-Only Read-Only RecordSet
Does not support scrolling or updating
Supports Databinding to Web controls
05.13

DataReader Object

When a large amount of data is being retrieved, maintaining a memory location for the data can become an issue. For example, reading 10,000 rows out of a database causes a DataTable to allocate and maintain memory for those 10,000 rows for the table's lifetime. If 1,000 users do this against the same machine at the same time, memory usage becomes critical. Also, if you are unsure of the results of a given query and you need code to iterate through the stream of results, a DataTable is not ideal, as again, the entire steam would be loaded.

To resolve this, you can use the **DataReader** which is a forward-only and read-only stream of data returned from the database. Only one record at a time is maintained in the memory. Because its functionality is very specific, it is "lightweight."

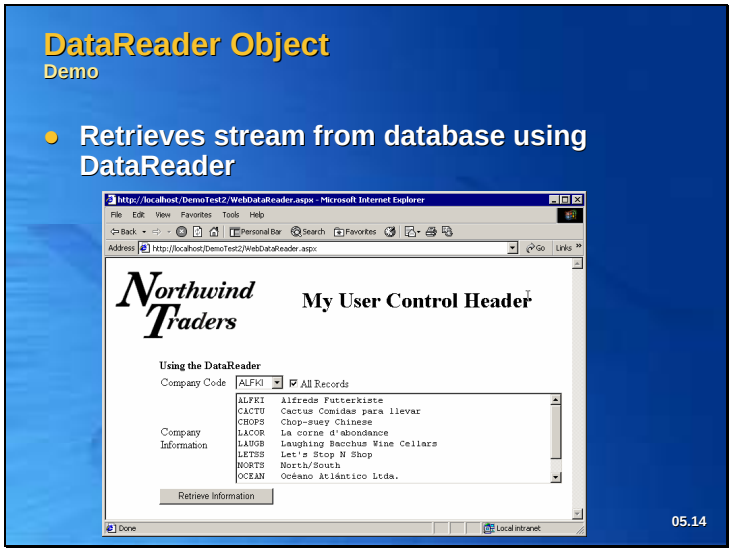
Instantiated by the OleDbCommand or SqlCommand class ExecuteReader method

You cannot actually instantiate the **OleDbDataReader** or **SqlDataReader** directly. DataReader objects can only be created by running the ExecuteReader method of a Command object.

Ties up the ADOCommand until it is finished reading

When using a DataReader to retrieve data, be aware that the **SqlCommand/OleDbCommand** class cannot process any other requests while it is reading data.

The DataReader can bind to Web controls but not Windows controls. This is because the Windows controls require an in-memory image of all the data and the DataReader stream contains only one record at a time.



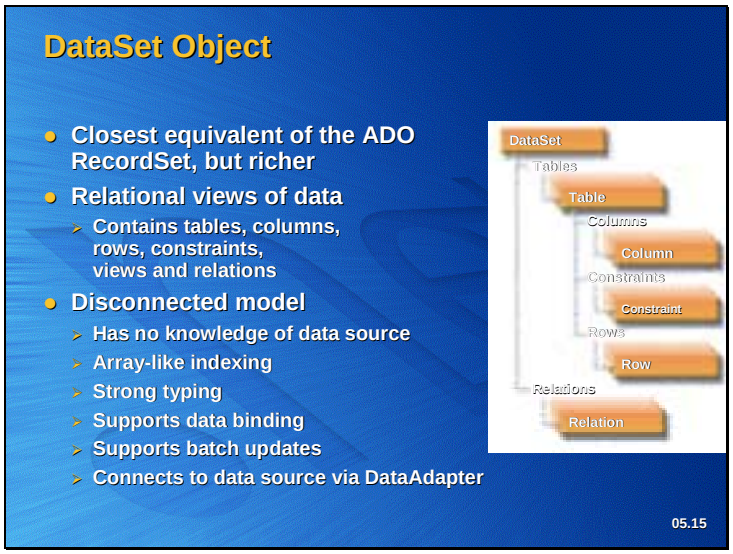
DataReader Object Demo

The purpose of this demo is to show you how to:

- Setup a Data Reader
- Interact with server controls
- Use server controls from server side code

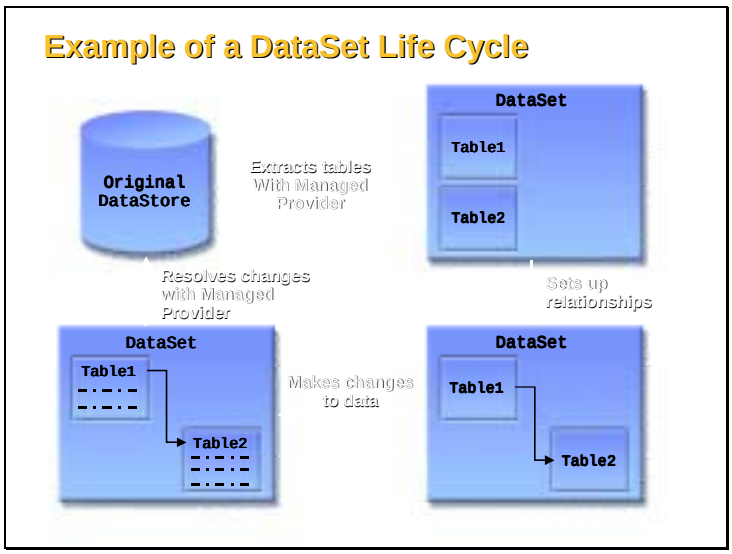
Note: The OleDb and the SqlClient objects have the same methods and properties. The only difference is the connection string you use.

 ©2002 Microsoft Corporation



DataSet Object

The centerpiece of any data access solution using ADO.NET is the DataSet. You can add data to a DataSet from several sources and then setup relationships between the data. A DataSet constitutes a "disconnected" view of the database data. That is, it exists in memory without an active connection to a database containing the corresponding tables or views. Once a DataSet has been filled, it does not know where the data came from. This disconnected architecture enables increased scalability by only using database server resources when reading or writing from the database.



Example of a DataSet Life Cycle

Populating Datasets

A DataSet is a container; therefore, you need to fill it with data. When you populate a DataSet, various events are raised; constraint checking applies, and so on. For details about updating a DataSet and the issues surrounding updates, see “Introduction to Data Updates in ADO.NET” in Visual Studio .NET Documentation.

You can populate a DataSet in a variety of ways:

- Call the **Fill** method of a data adapter.

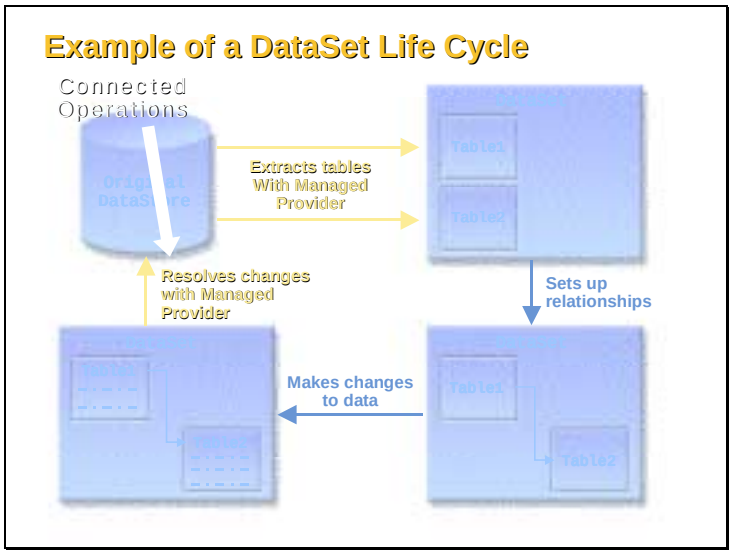
  This causes the adapter to execute a SQL statement or stored procedure and fill the results into a table in the DataSet. If the DataSet must contain data from multiple tables, you can create separate data adapters for each table, and call each adapter's **Fill** method separately. For details about how DataSets are filled, see “Introduction to Data Adapters” and “Creating Data Adapters” in Visual Studio .NET Documentation. For details on how to use a data adapter to populate a DataSet, see Populating a DataSet from a DataAdapter.

- Manually populate tables in the DataSet by creating **DataRow** objects and adding them to the table's **Rows** collection. (You can only do this at run time; you cannot set the **Rows** collection at design time.) For details, see Adding Data to a Table.

- Read an XML document or stream into the DataSet. For details, see the ReadXml method.

- Merge the contents of another DataSet. This scenario can be useful if your application gets DataSets from different sources (different Web Services, for example), but needs to consolidate them into a single DataSet. For details, see the DataSet.Merge method.

 ©2002 Microsoft Corporation

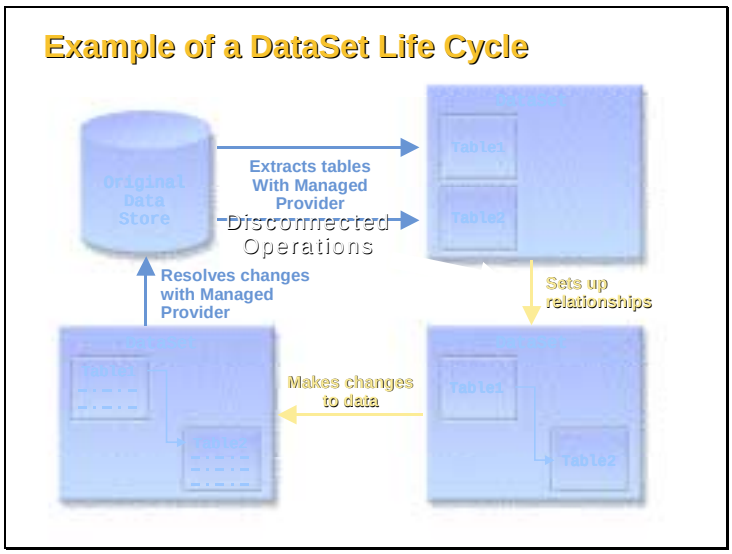


Example of a DataSet Life Cycle

Related Tables and DataRelation Objects

If you have multiple tables in a DataSet, the information in the tables can be related. A DataSet has no inherent knowledge of such relationships; to work with data in related tables, therefore, you can create DataRelation objects that describe the relations between the tables in the DataSet. **DataRelation** objects can be used to programmatically fetch related child records for a parent record, and a parent record from a child record.

You can use a **DataRelation** object to get related records from a child or parent table. For example, when working with the record describing the customer ANTON, you can get the collection of records describing the orders for that customer. Similarly, if you are working with the record describing order number 10507, you can use a **DataRelation** object to get the record describing the customer for that order (ANTON).

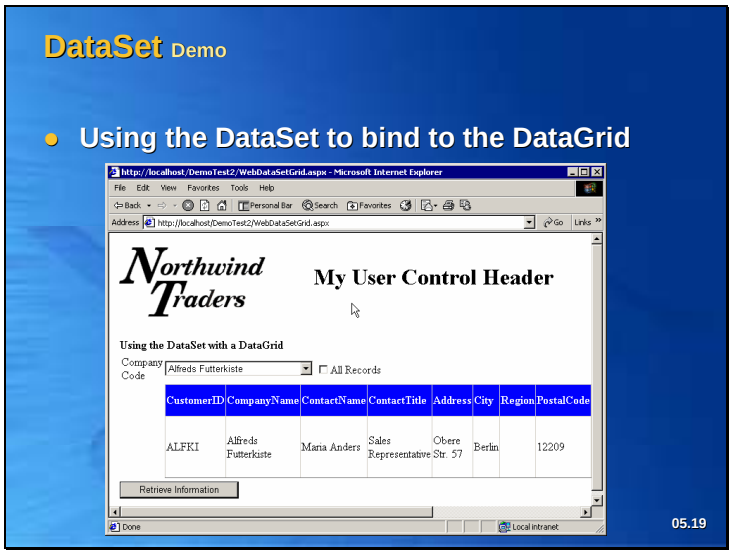


A DataSet Life Cycle

The life cycle of a **DataSet** starts when one of its constructor methods is called to initialize it . At the time of initialization the **DataSet** does not contain any data. The DataSet is populated by calling the **fill(DataSet dataSetObj,String dataTableName)** method of a **DataAdapter** class. The fill method opens the connection to the data source, if it is not already open, executes the SELECT statement as defined by the **SelectCommand** property of the DataAdapter, populates the **DataTable** in the **DataSet** object, and closes the connection to the data source. After the DataSet is populated, the relationships between the data tables within the DataSet can be defined. The DataSet can be bound with a DataGrid and changes be made to DataRows of **DataTable** within the DataSet. The setting up of relations and modifications to data happens in a disconnected mode whereas extraction of records from a data source to the DataSet happens in connected mode.

When the changes need to be applied from the DataSet to the data source, the **update (DataSet dataSetObj)** method is called on the **DataAdapter** that was earlier used to fill the DataSet. The update methods looks into the tables and rows of the DataSet to find out what changes have been made. The **DataRow** class used to manipulate individual records includes the **RowState** property, whose values indicate whether and how the row has been changed since the **DataTable** was first loaded from the database. Possible values include **Deleted**, **Modified**, **New**, and **Unchanged**. The **Update** method examines the value of the **RowState** property to determine which records need to be written to the database and what specific database command (insert, update or delete) should be invoked. It opens the connection to the data source, executes the DML statements and closes the connection.

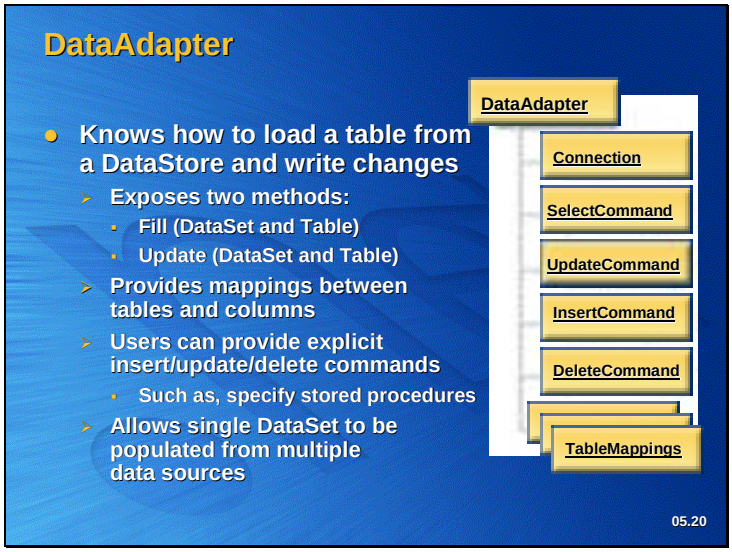
 ©2002 Microsoft Corporation



Data Set Demo

The purpose of this demo is to show you how to:

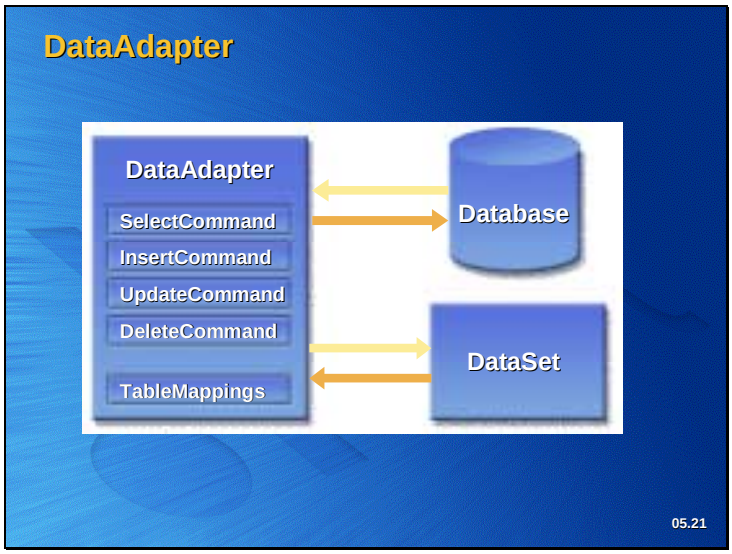
- Setup the DataSet
- Bind server controls to DataSet
- Use functions that return a DataSet



DataAdapter

The ADO.NET **DataSet** is a memory-resident representation of data that provides a consistent relational programming model independent of the data source. The **DataSet** represents a complete set of data including tables, constraints and relationships among the tables. Because the **DataSet** is independent of the data source, a **DataSet** can include data local to the application as well as data from multiple sources. Interaction with existing data sources is controlled through the **DataAdapter**.

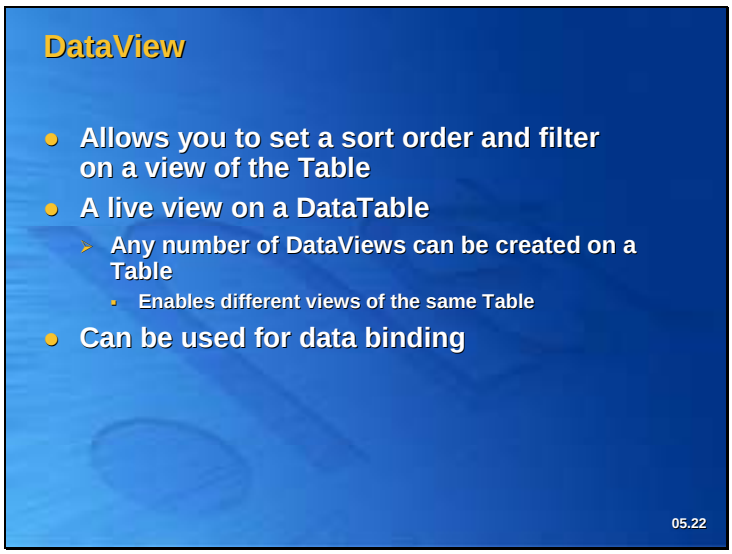
 ©2002 Microsoft Corporation



DataAdapter

The **DataAdapter** has four properties that retrieve data from the data source and update data at the data source using **Command** objects. The **SelectCommand** property returns data from the data source. There **DataAdapter** has three properties that are used to manage changes to the data source: the **InsertCommand**, **UpdateCommand** and **DeleteCommand**. The **SelectCommand** property must be set before calling the **Fill** method of the **DataAdapter**. The **InsertCommand**, **UpdateCommand** or **DeleteCommand** properties must be set before the **Update** method of the **DataAdapter** is called, depending on what changes were made to the data in the **DataSet**.

For example, if rows have been added, the **InsertCommand** must be set before calling **Update**. When **Update** is processing an inserted, updated or deleted row, the **DataAdapter** uses the respective **Command** property to process the action. Current information about the modified row is passed to the **Command** Object through the **Parameters** collection.


DataView
Allows you to set a sort order and filter on a view of the Table
A live view on a DataTable
Any number of DataViews can be created on a Table
Enables different views of the same Table
Can be used for data binding
05.22

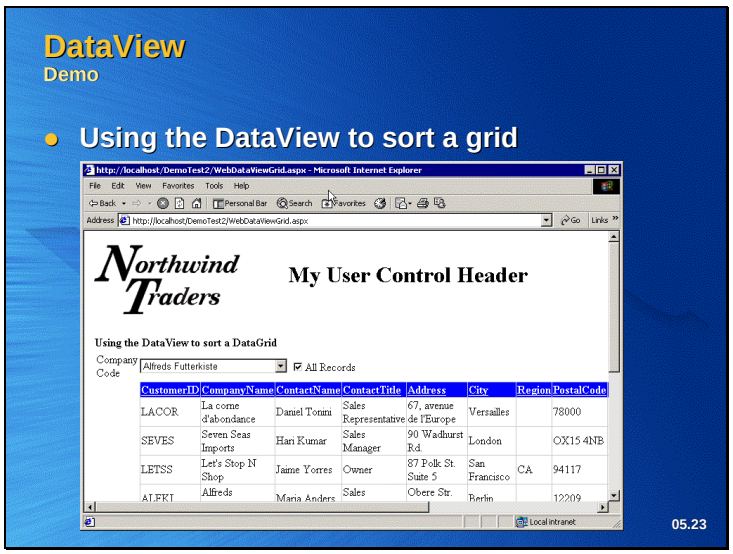
DataView

The **DataView Class** represents a data-bindable, customized view of a DataTable for sorting, filtering, searching, editing, and navigation. A DataView is similar to a live view on a DataTable, which allows programmers to set a sort order and filter on a view of the table. Any number of DataViews can be created to enable different views of the same table and used for data binding. However, a DataView is different from a database view. It can not be treated as a table and can not provide a view of joined tables. A DataView can not exclude columns of the source table or include additional columns such as computational columns.

A **DataView** allows programmers to set a default sort order and filter for individual tables. In addition, it allows DataViews to be linked and used for Databinding. Classically, data binding was used within applications to take advantage of data stored in databases. Windows Forms data binding allows you to access data from databases as well as data in other structures, such as arrays and collections (assuming certain minimum requirements have been met).

In Windows Forms, you can bind to a wide variety of structures, from simple (arrays) to complex (data rows, data views, and so on). As a minimum, a bindable structure must support the **IList** interface. As structures are based on increasingly capable interfaces, they offer more features that you can take advantage of when data binding. For a complete list of type of structures (data containers) you can bind to see the .Net Framework SDK Documentation on **IList** interface.

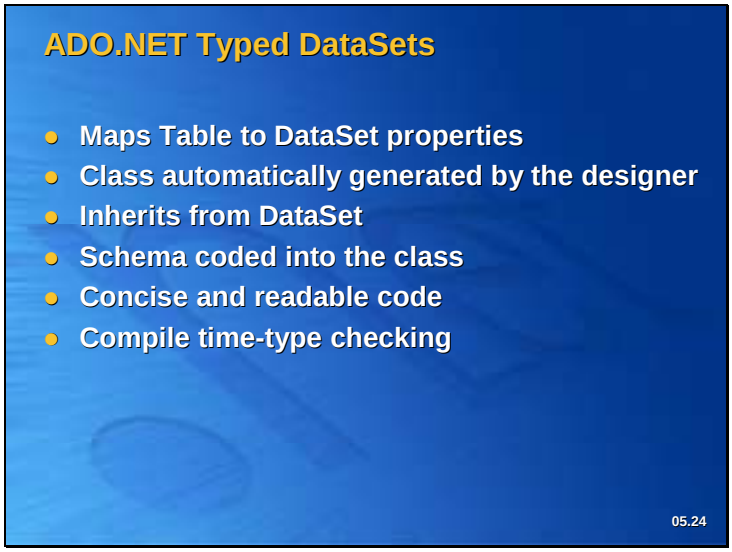
 ©2002 Microsoft Corporation


DataView
Demo
Using the DataView to sort a grid
http://localhost/DemoTest2/WebDataViewGrid.aspx - Microsoft Internet Explorer
Northwind Traders
My User Control Header
Using the DataView to sort a DataGrid
Company Code
Alfreds Futterkiste
All Records
CustomerID CompanyName ContactName ContactTitle Address City Region PostalCode
LACOR La corne d'abondance Daniel Tonini Sales Representative 67, avenue de l'Europe Versailles 78000
SEVES Seven Seas Imports Hari Kumar Sales Manager 90 Wadhurst Rd. London OX15 4NB
LETSS Let's Stop N Shop Jaime Yorres Owner 87 Polk St. Suite 5 San Francisco CA 94117
ALFKI Alfreds Maria Anders Sales Obere Str. Berlin 12209
Local intranet
05.23

Data View Demo

The purpose of this demo is to show how to:

- How sorting in the DataGrid works
- How to store DataSets in session variables and how it is serialized
- Flexibility of using data view and DataSets together



ADO.NET Typed DataSets

ADO.NET provides increased flexibility in DataSets and typed DataSets.

Typed DataSets versus Untyped Datasets

A typed DataSet is a class that is derived from the base **DataSet** class but includes type information as well. Typed DataSets are generated for schema information (tables, columns, and so on) stored in an XML schema file (an .xsd file). Information from the schema is generated and compiled into this new DataSet class as a set of first-class objects and properties.

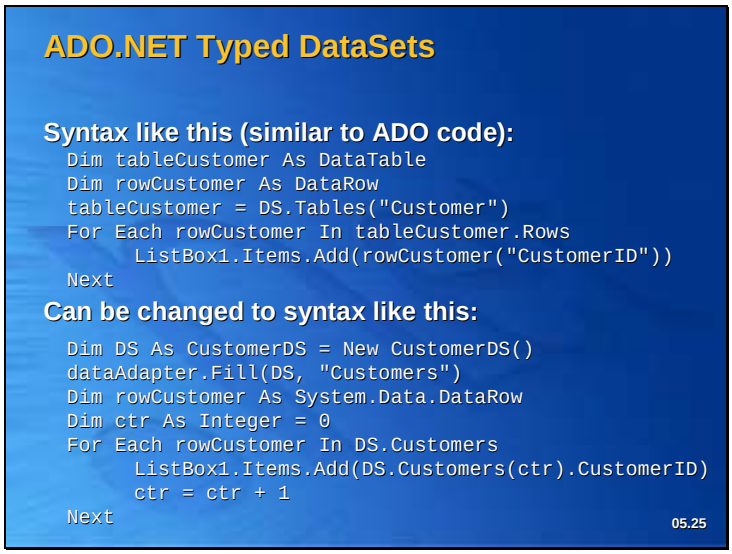
Note: For details about DataSet schemas, see “XML Data and Schemas” in Visual Studio .NET Documentation. Because a typed **DataSet** class is derived from the base **DataSet** class, the typed class inherits all of the functionality of the **DataSet** class and can be used with methods that take an instance of a **DataSet** class as a parameter

An untyped DataSet, in contrast, has no corresponding built-in schema. As in a typed DataSet, an untyped DataSet contains tables, columns, and so on — but these are only exposed as collections. However, after manually creating the tables and other data elements in an untyped DataSet, you can export the DataSet's structure as a schema using the DataSet's WriteXmlSchema method.

Typed DataSet enforces compile time type checking. This ensures that errors that may occur at runtime are handled at the design time, thereby improving the performance, security and type-safety.

You can use either type of DataSet in your applications. However, Visual Studio has more tool support for typed DataSets, that make programming with the DataSet easier and less error-prone.

 ©2002 Microsoft Corporation



ADO.NET Typed DataSets

Contrasting Data Access in Typed and Untyped Datasets

The following code accesses the CustomerID column in the first row of the Customers table.

Visual Basic Example

```
Dim CustID As String
CustID = CDS.Customers(0).CustomerID
```

C# Example

```
string CustID;
CustID = DS.Customers[0].CustomerID;
```

In contrast, if you are working with an untyped DataSet, the equivalent code is:

Visual Basic

```
Dim CustID As String
CustID =
CType(dsCustomersOrders1.Tables("Customers").Rows(0).Item("CustomerID")
, String)
```

C#

```
string CustID = (string)
dsCustomersOrders1.Tables["Customers"].Rows[0]["CustomerID"];
```

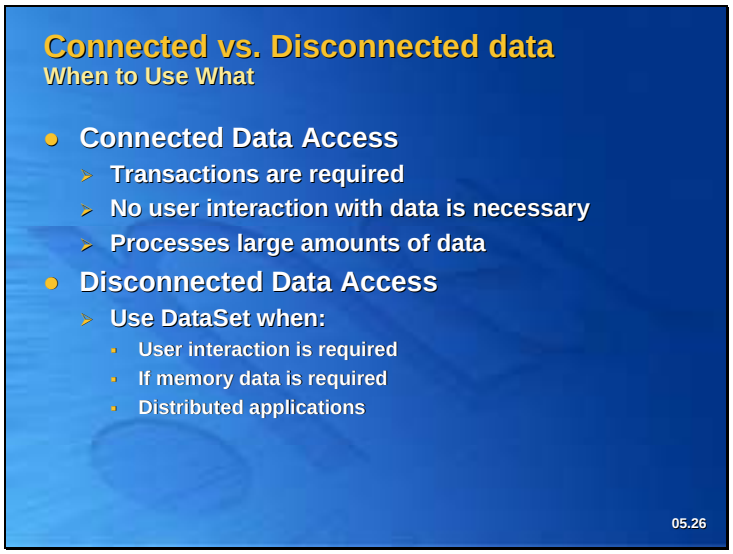
Typed access is not only easier to read, but is fully supported by IntelliSense in the Visual Studio Code Editor. In addition to being easier to work with, the syntax for the typed DataSet provides type checking at compile time, greatly reducing the possibility of errors in assigning values to DataSet members. Access to

tables and columns in a typed DataSet is also slightly faster at run time because access is determined at compile time, not through collections at run time.

Even though typed DataSets have many advantages, there are a variety of circumstances under which an untyped DataSet is useful. The most obvious scenario is that no schema is available for the DataSet. This might occur, for example, if your application is interacting with a component that returns a DataSet, but you do not know in advance what its structure is. Similarly, there are times when you are working with data that does not have a static, predictable structure; in that case, it is impractical to use a typed DataSet, because you would have to regenerate the typed DataSet class with each change in the data structure.

More generally, there are many times when you might create a DataSet dynamically without having a schema available. In that case, the DataSet is simply a convenient structure in which you can keep information, as long as the data can be represented in a relational way. At the same time, you can take advantage of the DataSet's capabilities, such as the ability to serialize the information to pass to another process, or to write out an XML file.

 ©2002 Microsoft Corporation



Connected vs. Disconnected Data

When to Use What

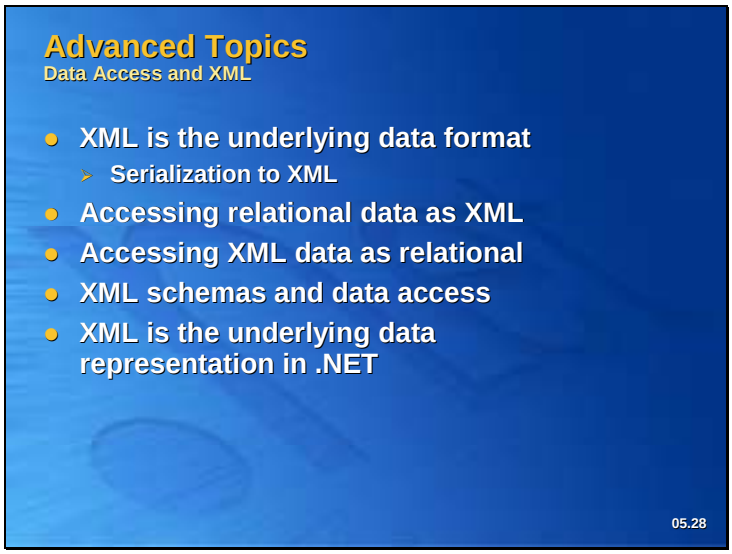
Data processing has traditionally relied on a connection-based, two-tier model. As data processing increasingly uses multi-tier architectures, programmers are switching to a disconnected approach to provide better scalability for their applications.

Summary

ADO.NET is a powerful successor to ADO, providing you with the same disconnected philosophy as the Web. Its architecture is flexible, allowing you represent your data in any logical way you wish. Furthermore, ADO.NET's usage of XML for representing data carries forward the .NET philosophy and ensures that data can be communicated with a wide variety of data sources, objects, and applications.

The data binding technologies for Visual Studio .NET have been improved to take full advantage of ADO.NET. Building user interfaces that interact with data is now easier, and more importantly, you now can bind values to business objects and XML Web services.

 ©2002 Microsoft Corporation


Advanced Topics
Data Access and XML
XML is the underlying data format
Serialization to XML
Accessing relational data as XML
Accessing XML data as relational
XML schemas and data access
XML is the underlying data representation in .NET
05.28

Advanced Topics

Data Access and XML

Getting relational data as XML

When you query a relational database such as SQL Server, the data can be loaded in a DataSet. The data in this DataSet can be accessed relationally as rows and columns or as an XML document. When you read data from a database table and put it in a DataSet, a DataTable is built. Each field in the query becomes a column in the DataSet and an element in the generated XML.

If you want to save a Dataset to XML, you can use the WriteXML, WriteXMLData, WriteXMLSchema methods to write the XML to an XMLWriter, a stream or a string.

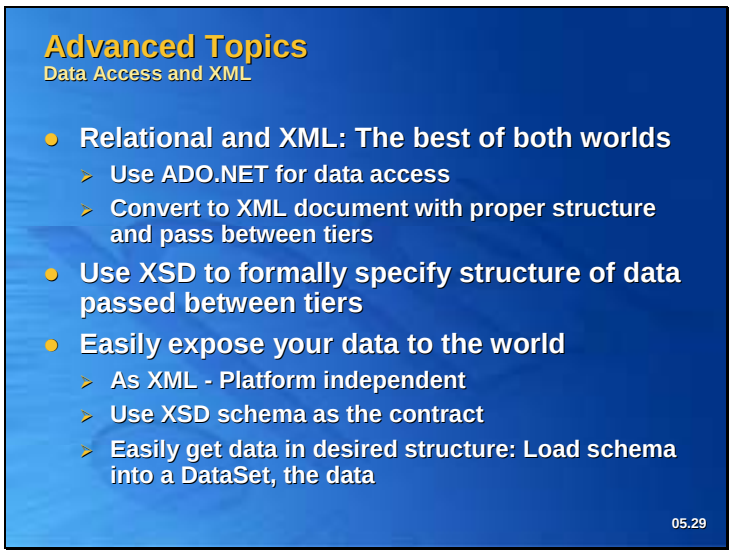
Show DataSetAsXML

You can also load the XML into a DOM Document and manipulate the document. This makes it easy to switch from the relational to the hierarchical programming model and to use the powerful XML features such as XPath to find data in the RecordSet. This is useful because a DataSet is like a mini-database in memory so you need a way to query this database.

Getting XML data as relational

Similarly, you can also load a DataSet from an XML document. This means you can use the relational API to manipulate XML documents rather than use the DOM. This is useful because you can create your business logic components to work with DataSets. You can then pass DataSets that contain data coming from XML without changing your business logic component design.

In fact, the integration is so tight that you can even bind controls to XML data. For example, you can create a grid that is bound to a DataSet which loads the data from XML. It then can save the data back to the XML document. **Show BindingToXML**


Advanced Topics
Data Access and XML
Relational and XML: The best of both worlds
Use ADO.NET for data access
Convert to XML document with proper structure and pass between tiers
Use XSD to formally specify structure of data passed between tiers
Easily expose your data to the world
As XML - Platform independent
Use XSD schema as the contract
Easily get data in desired structure: Load schema into a DataSet, the data
05.29

Advanced Topics

Data Access and XML

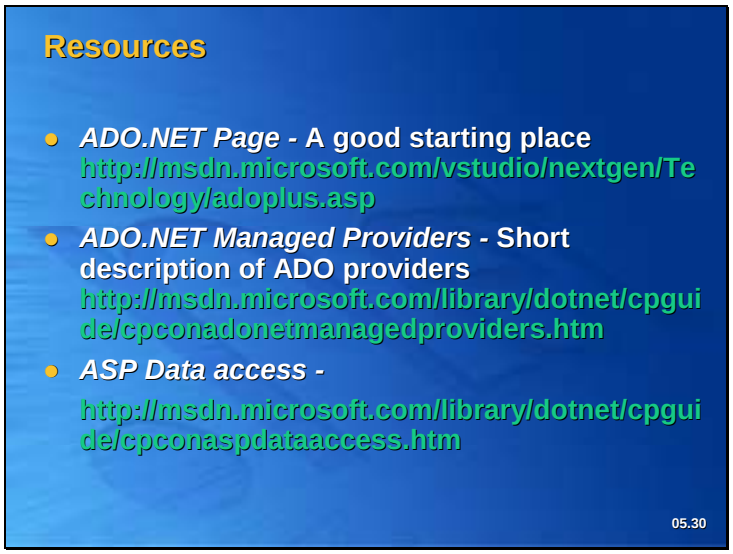
What this means to your application:

Distributed applications can use ADO.NET to get to the data and then easily convert this data to XML to propagate to other application layers.

Also, today, when you build a distributed application that passes RecordSets between tiers, you have no way of formally specifying the RecordSet schema. It is easy to make changes to the RecordSet that breaks the application, but those changes would not be caught at compile-time. With .NET, because of XSD and typed DataSets, you can use the XSD as the formal specification of the DataSet schema. All tiers can use typed DataSets based on this one schema. If the schema changes, those tiers would get compile-time errors and it would be easy to catch these problems early.

Also, because the ADO.NET data model is based on XML, you can easily expose data as XML to non-.NET applications. For example, if you are building an XML Web service accessible by Java or other languages, you can still use a DataSet. You simply need to create an XSD schema that describes the XML you want to expose as your Web service and then load that schema into your DataSet before you load the data from the database. When you serialize that DataSet to XML, it will do so according to the schema you created so your XML Web service clients get the XML they expect without knowing you are using a DataSet.

 ©2002 Microsoft Corporation

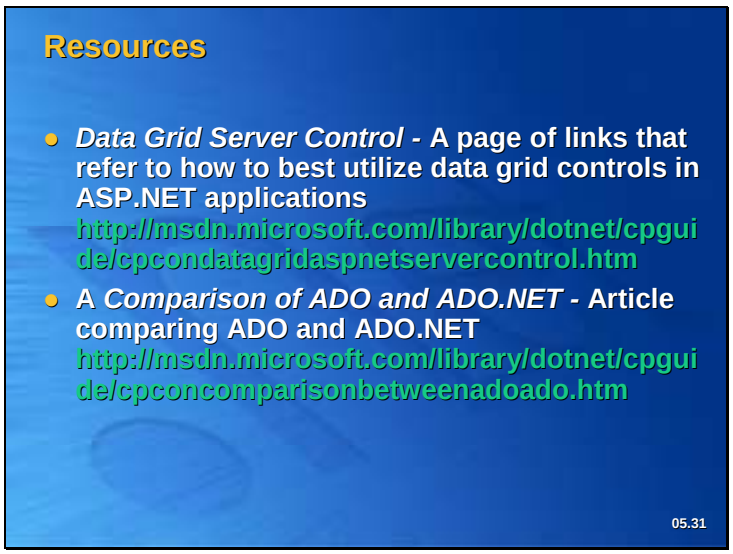


Resources

ADO.NET Page - A good starting place
http://msdn.microsoft.com/vstudio/nextgen/Technology/adoplus.asp

ADO.NET Managed Providers - Short description of ADO providers
http://msdn.microsoft.com/library/dotnet/cpguide/cpconadonetmanagedproviders.htm

ASP Data access - http://msdn.microsoft.com/library/dotnet/cpguide/cpconaspdataaccess.htm

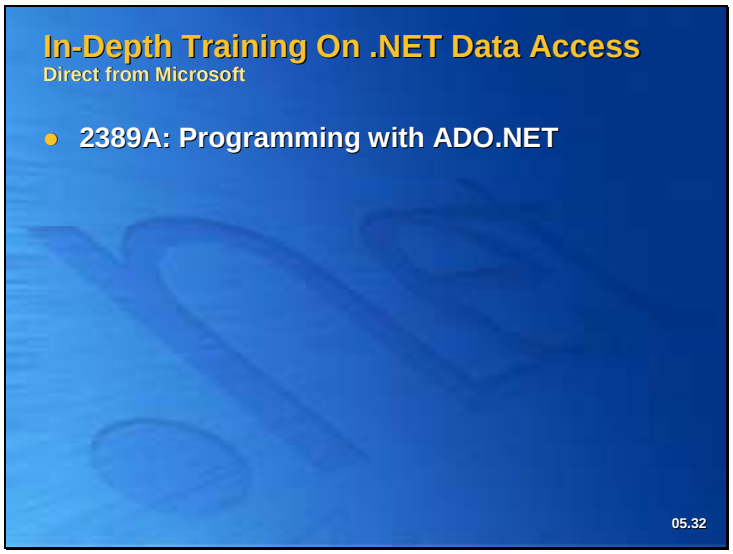

Resources
Data Grid Server Control - A page of links that refer to how to best utilize data grid controls in ASP.NET applications
http://msdn.microsoft.com/library/dotnet/cpguide/cpcondatagridaspnetservercontrol.htm
A Comparison of ADO and ADO.NET - Article comparing ADO and ADO.NET
http://msdn.microsoft.com/library/dotnet/cpguide/cpconcomparisonbetweenadoado.htm
05.31

Resources

Data Grid Server Control - A page of links that refer to how to best utilize data grid controls in ASP.NET applications http://msdn.microsoft.com/library/dotnet/cpguide/cpcondatagridaspnetservercontrol.htm

A *Comparison of ADO and ADO.NET* - Article comparing ADO and ADO.NET http://msdn.microsoft.com/library/dotnet/cpguide/cpconcomparisonbetweenadoado.htm

 ©2002 Microsoft Corporation



In-Depth Training On .NET Data Access

Direct from Microsoft

 ©2002 Microsoft Corporation